Math. Struct. in Comp. Science (2015), *vol.* 25, *pp.* 1692–1704. © Cambridge University Press 2014
doi:10.1017/S0960129513000327 First published online 10 November 2014

Program extraction in exact real arithmetic†

KENJI MIYAMOTO and HELMUT SCHWICHTENBERG

Mathematisches Institut, LMU, Theresienstr. 39, D-80333 München, Germany
Email: miyamoto@math.lmu.de, schwicht@math.lmu.de
Dedicated to John Tucker on occasion of his 60th birthday

Received 15 January 2013; revised 17 September 2014

The importance of an abstract approach to a computation theory over general data types has been stressed by Tucker in many of his papers. Berger and Seisenberger recently elaborated the idea for extraction out of proofs involving (only) abstract reals. They considered a proof involving coinduction of the proposition that any two reals in $[-1, 1]$ have their average in the same interval, and informally extract a Haskell program from this proof, which works with stream representations of reals. Here we formalize the proof, and machine extract its computational content using the Minlog proof assistant. This required an extension of this system to also take coinduction into account.

Extraction of programs from constructive proofs has received extensive interest in recent years (Kohlenbach 2008; Schwichtenberg and Wainer 2012). Here, we concentrate on an application area of particular importance, namely exact real arithmetic, seen as a subfield of constructive analysis. A standard way to approach the subject is to base the study on a concrete representation of (constructive) real numbers, like Cauchy sequences with a modulus (O'Connor 2009; Schwichtenberg 2008). However, an attractive alternative to such an approach has been proposed by Berger and Seisenberger (2010): one might start out with an *abstract* theory of reals instead. But then the immediate question is: how could one associate computational content with a formula $\forall_x \ldots$ where x ranges over abstract reals? By the very idea of abstractness we do (and should) not know the type of x; in fact, it may be a type variable. Therefore, the quantifier $\forall_x$ should have no computational significance. Following Berger (2009) we use a 'non-computational' universal quantifier $\forall_x^{\mathrm{nc}}$ instead, and move the computational content of our assumption into a predicate, i.e. we consider $\forall_x^{\mathrm{nc}}(Px \to \ldots)$ instead. This leaves the exact form of realizers for Px and hence the data type we use for representing real numbers open. Particularly relevant for concrete computations with exact real numbers is a stream representation based on signed digits, say $-1, 0, 1$. This has been realized very early: the thesis of Wiedmer (1977, 1980) (supervised by Engeler) is an example, and it is noted there that already Cauchy saw the usefulness of such a representation. Now, how can we achieve that Px has a stream of signed digits as realizer? An obvious way is to define the predicate P coinductively.

More precisely, we start out with a (free) algebra $\mathbf{I}$ of 'standard intervals', given by a nullary constructor $\mathbb{I}$ (representing the interval $[-1, 1]$) and a binary constructor C of

† Kenji Miyamoto is supported by the Marie Curie Initial Training Network in Mathematical Logic – MALOA – from Mathematical Logic to Applications, PITN-GA-2009-238381

https://doi.org/10.1017/S0960129513000327 Published online by Cambridge University Press

type $\mathbf{SD} \to \mathbf{I} \to \mathbf{I}$. Here $\mathbf{SD}$ are the 'signed digits' $-1, 0, 1$ (or $\mathtt{L}, \mathtt{M}, \mathtt{R}$ for left, middle, right); we write $\mathrm{C}_d v$ for $\mathrm{C}dv$. The intuition is that $\mathrm{C}_{d_0}(\mathrm{C}_{d_1} \dots (\mathrm{C}_{d_{k-1}} \mathbb{I}) \dots)$ denotes the interval in $[-1, 1]$ all of whose reals have a signed digit representation starting with $d_0 d_1 \dots d_{k-1}$. For example, $\mathrm{C}_1 \mathbb{I}$ denotes $[0, 1]$, $\mathrm{C}_0 \mathbb{I}$ denotes $[-\frac{1}{2}, \frac{1}{2}]$ and $\mathrm{C}_0(\mathrm{C}_{-1} \mathbb{I})$ denotes $[-\frac{1}{2}, 0]$. Generally, $\mathrm{C}_{d_0}(\mathrm{C}_{d_1} \dots (\mathrm{C}_{d_{k-1}} \mathbb{I}) \dots)$ denotes the interval with end points $\frac{1}{2^k}(\sum_{i<k} d_i 2^{k-1-i} \pm 1)$.

Now define inductively a set I of (abstract) reals, by the clauses

$$I0, \qquad \forall_x^{\mathrm{nc}} \forall_d \left(Ix \to I \frac{x+d}{2} \right). \tag{1}$$

Clearly, a witness for a proposition Ir according to the two clauses above can be seen as a constructor expression in our algebra $\mathbf{I}$, or alternatively as a 'total ideal' in the intended model (the base domain of the Scott–Ershov model of partial continuous functionals over free algebras; cf. Section 1.1 or Schwichtenberg and Wainer (2012) for details). For I we can (as for every inductively defined predicate) define its 'companion', the coinductively defined predicate ${}^{\mathrm{co}}I$, by the (single) clause

$$\forall_x^{\mathrm{nc}} \left({}^{\mathrm{co}}Ix \to x = 0 \vee \exists_y^{\mathrm{r}} \exists_d \left(x = \frac{y+d}{2} \wedge {}^{\mathrm{co}}Iy \right) \right) \tag{2}$$

($\exists_y^{\mathrm{r}}$ indicates that the existentially quantified variable y is disregarded in the realizability interpretation; cf. Section 1.2). A witness for a proposition ${}^{\mathrm{co}}Ir$ according to this clause is a finite or infinite stream of signed digits, indicating which signed digit d has been chosen in the second disjunct, and stopping if the first disjunct is taken. Such objects can be seen as 'cototal ideals' for the algebra $\mathbf{I}$ (cf. Section 1.1). Now we can formulate the proposition to be proved:

$$\forall_{x,y}^{\mathrm{nc}} \left({}^{\mathrm{co}}Ix \to {}^{\mathrm{co}}Iy \to {}^{\mathrm{co}}I \frac{x+y}{2} \right), \tag{3}$$

where addition $+$ and division by 2 are performed on the abstract level; in fact, x, y are ignored by the realizability interpretation, because of the non-computational $\forall_{x,y}^{\mathrm{nc}}$. The only computational effect of addition $+$ and division by 2 is the way in which their definition influences the proof of (3). By the above the computational content of (3) will be a stream transformer, and this is exactly what we want.

Now, what will be special in the proof of (3)? To reasonably work with the predicate ${}^{\mathrm{co}}I$ we will need coinduction, or more precisely, the greatest-fixed-point axiom for ${}^{\mathrm{co}}I$. On the level of extracted terms we will have to provide the corecursion operator, as the computational content of coinduction.

In Berger and Seisenberger (2010) the idea for extraction out of proofs involving (only) abstract reals is presented. As a case study, they consider a proof of proposition (3) above about the average function. Based on an informal understanding of the computational content of this proof, they write down a Haskell program expressing this content. At the end of Berger and Seisenberger (2010) a desire for an automation of such an extraction process is expressed. The present paper reports on how this can be achieved using the proof assistant Minlog[†], and illustrates it with the case study at hand. We found it

[†] See `http://www.minlog-system.de/`

https://doi.org/10.1017/S0960129513000327 Published online by Cambridge University Press

helpful to take TCF ('theory of computable functionals', cf. Section 1 or Schwichtenberg and Wainer (2012)) as the underlying theory. This made it possible to directly formalize the informal proof (cf. Section 2). However, it was necessary to extend Minlog by (i) coinductively defined predicates, (ii) their greatest-fixed-point axioms (i.e. coinduction) and (iii) corecursion operators as realizers for the coinduction axioms. Moreover, we had to deal with the problem that terms involving corecursion operators do not have a normal form. Using such an extension of Minlog we could automatically extract a term realizing proposition (3) from its proof (Section 3.2), and test it on concrete input data (Section 3.4).

An important aspect of this way of dealing with extraction of computational content from proofs is that what is extracted is a term in the language of the underlying theory. This makes it possible to prove (again in the formal theory) that this term is indeed a realizer of its specification (3) (Section 3.3).

Comparison with the literature

The importance of an abstract approach to a computation theory over general data types has been stressed by Tucker in many of his papers (for instance Tucker and Zucker (1992)). The present paper is based heavily on the paper of Berger and Seisenberger (2010), as explained above. The average function has been studied from an exact real number perspective by Plume (1998) and in Ciaffaglione and Gianantonio (2006). The latter paper starts with explicit definitions of stream transformers; correctness is verified in the proof assistant Coq (Coq Development Team 2009). Work involving coinduction is also prominent in the proof assistant Agda (2013); see the recent Ph.D. thesis of Chuang (2011). However, in Agda whole proofs are taken as programs (not only their computationally relevant parts), and this causes difficulties (special care is needed to guarantee that postulates do not prevent normalization).

1. The formal system

We discuss particular features of the underlying formal system TCF ('theory of computable functionals', cf. Schwichtenberg and Wainer (2012)), which are relevant for the example at hand.

1.1. *Algebras and their total and cototal ideals*

Rather than working with algebras and coalgebras in a categorical setting (as for instance done in Berger and Seisenberger (2010)), we just use (free) algebras to generate the basic domains of the Scott–Ershov model of partial continuous functionals. Among the ideals of such a domain we single out the total and cototal ones, which are our well-founded and non-well-founded objects, respectively. We construct domains by information systems, given by a set of tokens, a set of finite consistent sets of tokens, and an entailment relation. As an example, consider the algebra **I** of standard intervals introduced above, and let C range over its constructors. The following definitions are an adaption of the more general ones in Schwichtenberg and Wainer (2012) to the present case.

https://doi.org/10.1017/S0960129513000327 Published online by Cambridge University Press

a. The *tokens* a are the type correct constructor expressions (or trees) $\mathrm{C}a_1^* \dots a_n^*$ where a_i^* is an *extended token*, i.e. a token or the special symbol $*$ which carries no information.
b. A finite set U of tokens is *consistent* if all its elements start with the same constructor C, say of arity $\tau_1 \to \dots \to \tau_n \to \mathbf{I}$, and all U_i are consistent ($i = 1, \dots, n$), where U_i consists of all (proper) tokens at the ith argument position of some token in $U = \{\mathrm{C}\vec{a_1^*}, \dots, \mathrm{C}\vec{a_m^*}\}$.
c. $\{\mathrm{C}\vec{a_1^*}, \dots, \mathrm{C}\vec{a_m^*}\} \vdash \mathrm{C}'\vec{a^*}$ ('entails') is defined to mean $\mathrm{C} = \mathrm{C}'$, $m \geqslant 1$ and $U_i \vdash a_i^*$, with U_i as in (b) above (and $U \vdash *$ taken to be true).

These are definitions by recursion on the height (of the syntactic expressions involved), defined by

$$|\mathrm{C}a_1^* \dots a_n^*| := 1 + \max\{\, |a_i^*| \mid i = 1, \dots, n \,\}, \qquad |*| := 0,$$
$$|\{\, a_i \mid i \in I \,\}| := \max\{\, 1 + |a_i| \mid i \in I \,\},$$
$$|U \vdash a| := \max\{1 + |U|, 1 + |a|\}.$$

A set of tokens is *deductively closed* if it contains all tokens entailed from one of its finite subsets. An *ideal* is defined to be a (possibly infinite) consistent and deductively closed set of tokens. The intuition is that a finite consistent set U of tokens is seen as a 'formal neighbourhood' (Kreisel 1959) in a space of abstract points or ideals.

To define total and cototal ideals, consider a constructor tree $P(*)$ with a distinguished occurrence of $*$. Writing $\mathrm{C}_d^n a$ for $\mathrm{C}_d(\mathrm{C}_d(\dots(\mathrm{C}_d\, a)\dots))$, for $P(*) := \mathrm{C}_{-1}\mathrm{C}_1^2 *$ we have $P(\mathrm{C}_1 *) = \mathrm{C}_{-1}\mathrm{C}_1^3 * \succ_1 \mathrm{C}_{-1}\mathrm{C}_1^2 * = P(*)$. Then an arbitrary $P(\mathrm{C}\vec{a^*})$ is called *one-step extension* of $P(*)$, written $P(\mathrm{C}\vec{a^*}) \succ_1 P(*)$. An ideal x is called *cototal* if every constructor expression $P(*) \in x$ has a one-step extension $P(\mathrm{C}\vec{a^*}) \in x$; it is called *total* if it is cototal and the relation $\succ_1$ on x is well founded. Every total ideal then can be seen as a standard interval

$$\mathbb{I}_{i \cdot 2^{-k}, k} := \left[\frac{i}{2^k} - \frac{1}{2^k}, \frac{i}{2^k} + \frac{1}{2^k}\right] \qquad \text{for } -2^k < i < 2^k.$$

The cototal ideals are what we mean by a 'stream representation' of reals. For instance, the cototal ideals include $\{\, \mathrm{C}_{-1}^n * \mid n \geqslant 0 \,\}$, a stream representation of the real -1, and also $\{\, \mathrm{C}_1\mathrm{C}_{-1}^n * \mid n \geqslant 0 \,\}$ and $\{\, \mathrm{C}_{-1}\mathrm{C}_1^n * \mid n \geqslant 0 \,\}$, which both represent the real 0. Generally, the cototal ideals give us all reals in $[-1, 1]$, in the (non-unique) stream representation via signed digits $-1, 0, 1$.

1.2. *Realizability*

We now address the issue of extracting computational content from proofs. The method of program extraction is based on *modified realizability* as introduced by Kreisel (1959) and described in detail in Schwichtenberg and Wainer (2012). In short, from every constructive proof M of a non-Harrop formula A (in natural deduction) one extracts a program $\mathrm{et}(M)$ 'realizing' A, essentially by removing computationally irrelevant parts from the proof (proofs of Harrop formulas have no computational content). The extracted program has some simple type $\tau(A)$ which depends solely on the logical shape of the proven formula A. In its original form the extraction process is fairly straightforward, but often leads to unnecessarily complex programs. In order to obtain better programs,

https://doi.org/10.1017/S0960129513000327 Published online by Cambridge University Press

proof assistants (for instance Coq, Isabelle/HOL, Agda, Nuprl, Minlog) offer various optimizations of program extraction. Below we describe optimizations implemented in Minlog (Schwichtenberg 2006), which are relevant for our present case study.

Quantifiers without computational content. Besides the usual quantifiers, $\forall$ and $\exists$, Minlog has so-called *non-computational quantifiers*, $\forall^{\mathrm{nc}}$ and $\exists^{\mathrm{r}}$, which allow for the extraction of simpler programs. These quantifiers, which were first introduced by Berger (1993), can be viewed as a refinement of the Set/Prop distinction in constructive type systems like Coq or Agda. Intuitively, a proof of $\forall^{\mathrm{nc}}_x A(x)$ ($A(x)$ non-Harrop) represents a procedure that assigns to any x a proof $M(x)$ of $A(x)$ where $M(x)$ does not make 'computational use' of x, i.e. the extracted program $\mathrm{et}(M(x))$ does not depend on x. Dually, a proof of $\exists^{\mathrm{r}}_x A(x)$ is a proof of $M(x)$ for some x where the witness x is 'hidden', that is, not available for computational use (the 'r' stands for 'right'); in fact, $\exists^{\mathrm{r}}$ can be seen as inductively defined by the clause $\forall^{\mathrm{nc}}_x(A \to \exists^{\mathrm{r}}_x A)$. The types of extracted programs for non-computational quantifiers are $\tau(\forall^{\mathrm{nc}}_{x^\rho} A) = \tau(\exists^{\mathrm{r}}_{x^\rho} A) = \tau(A)$ as opposed to $\tau(\forall_{x^\rho} A) = \rho \to \tau(A)$ and $\tau(\exists_{x^\rho} A) = \rho \times \tau(A)$. The extraction rules are, for example in the case of $\forall^{\mathrm{nc}}$-introduction and -elimination, $\mathrm{et}((\lambda_x M^{A(x)})^{\forall^{\mathrm{nc}}_x A(x)}) = \mathrm{et}(M)$ and $\mathrm{et}((M^{\forall^{\mathrm{nc}}_x A(x)} t)^{A(t)}) = \mathrm{et}(M)$ as opposed to $\mathrm{et}((\lambda_x M^{A(x)})^{\forall_x A(x)}) = \mathrm{et}(\lambda_x M)$ and $\mathrm{et}((M^{\forall_x A(x)} t)^{A(t)}) = \mathrm{et}(Mt)$. For the extracted programs to be correct the variable condition at $\forall^{\mathrm{nc}}$-introduction must be strengthened by requiring in addition the abstracted variable x not to occur in the extracted program $\mathrm{et}(M)$, and similarly for $\exists^{\mathrm{r}}$. Note that for a Harrop formula A the formulas $\forall^{\mathrm{nc}}_x A$, $\forall_x A$ and also $\exists^{\mathrm{r}}_x A$, $\exists_x A$ are equivalent.

Animation. Suppose a proof of a theorem uses a lemma. Then the proof term contains just the name of the lemma, say `L`. In the term extracted from this proof we want to preserve the structure of the original proof as much as possible, and hence use a new constant `cL` at those places where the computational content of the lemma is needed. When we want to execute the program, we have to replace the constant `cL` corresponding to a lemma `L` by the extracted program of its proof. This can be achieved by adding computation rules for `cL`. We can be rather flexible here and enable/block rewriting by using `animate`/`deanimate` as desired.

1.3. *Inductive and coinductive definitions*

To be able to work from a computational point of view with our abstract reals, we have to inductively define what we need to know in order to be able to view an abstract real as a computational object. This is done by means of inductive and coinductive definitions.

As an example, consider the inductive definition of I by the clauses (1) and the coinductive definition of ${}^{\mathrm{co}}I$ by (2). We already noted above that a witness for a proposition Ir can be seen as a total ideal in our algebra $\mathbf{I}$, and a witness for a proposition ${}^{\mathrm{co}}Ir$ as a cototal ideal. We still need to express that I is the least predicate satisfying the two clauses (1). This is done by means of the *least-fixed-point* axiom

$$\forall^{\mathrm{nc}}_x \left(Ix \to P0 \to \forall^{\mathrm{nc}}_y \forall_d \left(Iy \to Py \to P\frac{y+d}{2} \right) \to Px \right). \tag{4}$$

https://doi.org/10.1017/S0960129513000327 Published online by Cambridge University Press

Dually, we need to express that ${}^{\mathrm{co}}I$ is the greatest predicate satisfying (2), by the *greatest-fixed-point* axiom

$$\forall_x^{\mathrm{nc}}\left(Px \to \forall_y^{\mathrm{nc}}\left(Py \to y = 0 \vee \exists_d \exists_z^{\mathrm{r}}\left(y = \frac{z+d}{2} \wedge ({}^{\mathrm{co}}Iz \vee Pz)\right)\right) \to {}^{\mathrm{co}}Ix\right). \tag{5}$$

Both can be understood as dealing with a 'competitor' predicate P satisfying the same clauses/clause as $I/{}^{\mathrm{co}}I$. Then (4) says that P is a superset of I, and (5) that P is a subset of ${}^{\mathrm{co}}I$.

The computational content of these axioms depends on the type $\tau := \tau(Pr)$ of P. Then, the term extracted from the least-fixed-point axiom (4) is Gödel's (structural) recursion operator $\mathcal{R}_{\mathbf{I}}^{\tau}$, and the term extracted from the greatest-fixed-point axiom (5) is the 'corecursion' operator ${}^{\mathrm{co}}\mathcal{R}_{\mathbf{I}}^{\tau}$ defined below (in Section 1.4). Moreover, the terms extracted from the clauses (1) for I are the constructors of $\mathbf{I}$, and the term extracted from the clause (2) for ${}^{\mathrm{co}}I$ is the *destructor* $\mathcal{D}_{\mathbf{I}}$ of type $\mathbf{I} \to \mathbf{U} + \mathbf{SD} \times \mathbf{I}$ (where $\mathbf{U}$ is the unit type containing $\mathbf{u}$ only), defined by

$$\mathcal{D}(\mathbb{I}) := \mathrm{inl}\,\mathbf{u}, \qquad \mathcal{D}(\mathrm{C}_d v) := \mathrm{inr}\langle d, v\rangle.$$

Note that the application described in this paper does not require (5), but the following weaker axiom suffices.

$$\forall_x^{\mathrm{nc}}\left(Px \to \forall_y^{\mathrm{nc}}\left(Py \to y = 0 \vee \exists_d \exists_z^{\mathrm{r}}\left(y = \frac{z+d}{2} \wedge Pz\right)\right) \to {}^{\mathrm{co}}Ix\right).$$

However, using the stronger form (5) does not complicate matters, and it is the proper (general) way to state elimination axioms. A similar remark applies to Section 1.4.

1.4. *Corecursion*

Streams are infinite objects, and require a special treatment when computing with them. It is well known that an arbitrary 'reduction sequence' beginning with a term in Gödel's T terminates. For this to hold it is essential that the constants allowed in T are restricted to constructors C and recursion operators $\mathcal{R}$. A consequence is that every closed term of a base type denotes a total ideal. The conversion rules for $\mathcal{R}$ work from the leaves towards the root, and terminate because total ideals are well founded. If, however, we deal with cototal ideals (streams, for example), then a similar operator is available to define functions with cototal ideals as values, namely corecursion. The corecursion operator ${}^{\mathrm{co}}\mathcal{R}_{\mathbf{I}}^{\tau}$ is used to construct a mapping from τ to $\mathbf{I}$ by corecursion on the structure of $\mathbf{I}$. Its type and conversion relation are

$$\begin{aligned} &{}^{\mathrm{co}}\mathcal{R}_{\mathbf{I}}^{\tau} : \tau \to (\tau \to \mathbf{U} + \mathbf{SD} \times (\mathbf{I} + \tau)) \to \mathbf{I} \\ &{}^{\mathrm{co}}\mathcal{R}_{\mathbf{I}}^{\tau} NM \mapsto [\lambda_{_}\mathbb{I}, \lambda_{d,z}(\mathrm{C}_d([\lambda_{_}\mathbb{I}, \lambda_u({}^{\mathrm{co}}\mathcal{R}_{\mathbf{I}}^{\tau} uM)]z))](MN) \end{aligned}$$

where $[f, g] : \rho + \sigma \to \tau$ is defined for $f : \rho \to \tau$ and $g : \sigma \to \tau$ by

$$[f, g](\mathrm{inl}\, x^{\rho}) := f(x), \qquad [f, g](\mathrm{inr}\, y^{\sigma}) := g(y).$$

https://doi.org/10.1017/S0960129513000327 Published online by Cambridge University Press

We write an underscore for a variable whose name is irrelevant. In a more readable notation we have

$$
\begin{aligned}
{}^{\mathrm{co}}\mathcal{R}^{\tau}_{\mathbf{I}} NM \mapsto\ & [\mathbf{case}\ (MN)^{\mathbf{U}+\mathbf{SD}\times(\mathbf{I}+\tau)}\ \mathbf{of} \\
& \quad \mathrm{inl}\ _ \mapsto \mathbb{I}\ \mid \\
& \quad \mathrm{inr}\langle d, z\rangle \mapsto \mathrm{C}_d[\mathbf{case}\ z^{\mathbf{U}+\tau}\ \mathbf{of} \\
& \qquad\qquad \mathrm{inl}\ _ \mapsto \mathbb{I}\ \mid \\
& \qquad\qquad \mathrm{inr}\ u^{\tau} \mapsto {}^{\mathrm{co}}\mathcal{R}^{\tau}_{\mathbf{I}} uM]].
\end{aligned}
\tag{6}
$$

As an example of a function defined by corecursion consider the transformation of an abstract real in the interval $[-1, 1]$ into a stream representation using signed digits. Assume that we work in an abstract (axiomatic) theory of reals, having an unspecified type ρ, and that we have a type $\mathbf{Q}$ for rationals as well. Assume that the abstract theory provides us with a function $g : \rho \to \mathbf{Q} \to \mathbf{Q} \to \mathbf{B}$ ($\mathbf{B}$ is the type of booleans) comparing a real x with a proper rational interval $p < q$:

$$
\begin{aligned}
g(x, p, q) &= \mathtt{tt} \to x \leqslant q, \\
g(x, p, q) &= \mathtt{ff} \to p \leqslant x.
\end{aligned}
$$

From g we define a function $h : \rho \to \mathbf{U} + \mathbf{SD} \times (\mathbf{I} + \rho)$ by

$$
h(x) := \begin{cases}
\mathrm{inr}\langle -1, \mathrm{inr}(2x+1)\rangle & \text{if } g\left(x, -\tfrac{1}{2}, 0\right) = \mathtt{tt}, \\
\mathrm{inr}\langle 0, \mathrm{inr}(2x)\rangle & \text{if } g\left(x, -\tfrac{1}{2}, 0\right) = \mathtt{ff},\ g\left(x, 0, \tfrac{1}{2}\right) = \mathtt{tt}, \\
\mathrm{inr}\langle 1, \mathrm{inr}(2x-1)\rangle & \text{if } g\left(x, 0, \tfrac{1}{2}\right) = \mathtt{ff}.
\end{cases}
$$

h is definable by a closed term M in Gödel's T. Then, the desired function $f : \rho \to \mathbf{I}$ transforming an abstract real x into a cototal ideal (i.e. a stream) in $\mathbf{I}$ can be defined by

$$
f(x) := {}^{\mathrm{co}}\mathcal{R}^{\rho}_{\mathbf{I}} xM.
$$

This $f(x)$ will thus be a stream of signed digits $-1, 0, 1$.

2. The informal proof

We now present an informal proof of proposition (3) above about the average function, closely following Berger and Seisenberger (2010). This proof and in particular the notions involved in it will be formalized in Section 3 below. By convention on variable names, d, e are for $\mathbf{SD}$ and i, j are for $\mathbf{SD}_2 := \{-2, -1, 0, 1, 2\}$ (or LL, LT, MT, RT, RR), the algebra of 'extended signed digits'.

Theorem (Average).

$$
\forall^{\mathrm{nc}}_{x,y}\left({}^{\mathrm{co}}Ix \to {}^{\mathrm{co}}Iy \to {}^{\mathrm{co}}I\,\frac{x+y}{2}\right).
$$

Proof. Let

$$
X := \left\{\frac{x+y}{2} \mid x, y \in {}^{\mathrm{co}}I\right\}, \qquad Y := \left\{\frac{x+y+i}{4} \mid x, y \in {}^{\mathrm{co}}I, i \in \mathbf{SD}_2\right\}.
$$

https://doi.org/10.1017/S0960129513000327 Published online by Cambridge University Press

Below we will show $X \subseteq Y$ and that Y satisfies the clause coinductively defining ${}^{\mathrm{co}}I$. Therefore by the greatest-fixed-point for ${}^{\mathrm{co}}I$ we have $Y \subseteq {}^{\mathrm{co}}I$. Hence $X \subseteq {}^{\mathrm{co}}I$, which is our claim. □

In the following we abbreviate $\forall_x^{\mathrm{nc}}({}^{\mathrm{co}}I x \to A)$ by $\forall_{x \in {}^{\mathrm{co}}I}^{\mathrm{nc}} A$, and similarly for existential quantifiers.

Lemma (XSubY).

$$\forall_{x,y \in {}^{\mathrm{co}}I}^{\mathrm{nc}} \forall_z^{\mathrm{nc}} \left(z = \frac{x+y}{2} \to \exists_i \exists_{x',y' \in {}^{\mathrm{co}}I}^{\mathrm{r}}\ z = \frac{x'+y'+i}{4} \right).$$

Proof. Let $x, y \in {}^{\mathrm{co}}I$ and $z := \frac{x+y}{2}$. Assume for instance $x = \frac{x'+d}{2}$ and $y = \frac{y'+e}{2}$ for some $x', y' \in {}^{\mathrm{co}}I$ and $d, e \in \mathbf{SD}$. Then $z = \frac{x+y}{2} = \frac{x'+y'+d+e}{4}$. In case $x = 0$ and $y = \frac{y'+e}{2}$ we have $z = \frac{x+y}{2} = \frac{y'+e}{4}$. The other cases are similar. □

Lemma (YSatClause).

$$\forall_i \forall_{x,y \in {}^{\mathrm{co}}I}^{\mathrm{nc}} \forall_z^{\mathrm{nc}} \Bigg(z = \frac{x+y+i}{4} \to$$
$$z = 0 \vee \exists_{j,d} \exists_{x',y' \in {}^{\mathrm{co}}I}^{\mathrm{r}} \exists_{z'}^{\mathrm{r}} \left(z' = \frac{x'+y'+j}{4} \wedge z = \frac{z'+d}{2} \right) \Bigg).$$

Proof. Let $i \in \mathbf{SD}_2$ and $x, y \in {}^{\mathrm{co}}I$. We show that $z := \frac{x+y+i}{4}$ satisfies the right-hand side of the disjunction. In case $x = \frac{x'+d'}{2}$ and $y = \frac{y'+e'}{2}$ we have $z = \frac{x'+y'+d'+e'+2i}{8}$. Solve $d' + e' + 2i = j + 4d$. Then for $z' := \frac{x'+y'+j}{4}$

$$\frac{z'+d}{2} = \frac{4z'+4d}{8} = \frac{x'+y'+j+4d}{8} = \frac{x'+y'+d'+e'+2i}{8} = z.$$

The other cases are simpler. □

3. Formalization and extraction

Since the formal proof follows rather closely the informal one above, we do not comment on how it is generated interactively, but only on some of the more interesting points, and the extracted terms.

3.1. *Formalization*

Abstract reals. We use a type variable ρ to denote an abstract type of reals. To formulate their properties, we need functions `P` (plus) of type $\rho \to \rho \to \rho$ for addition, and `H` (half) of type $\rho \to \rho$ for division by 2. Moreover, we need auxiliary functions connecting the concrete data types $\mathbf{Z}$ (integers), $\mathbf{SD}$ (signed digits $\{-1, 0, 1\}$) and $\mathbf{SD}_2$ (extended signed digits $\{-2, -1, 0, 1, 2\}$) with our abstract reals and also among themselves. These are

$$\texttt{SDToInt} : \mathbf{SD} \to \mathbf{Z}, \qquad \texttt{SDtwoToInt} : \mathbf{SD}_2 \to \mathbf{Z}, \qquad \texttt{IntToR} : \mathbf{Z} \to \rho.$$

https://doi.org/10.1017/S0960129513000327 Published online by Cambridge University Press

We use $+$ for `P` possibly with implicit `IntToR`, and $+_{\mathbf{Z}}$ for integer addition. The properties we need to assume for our abstract reals are

$$
\begin{aligned}
&(x+k)/2+l=(x+(k+_{\mathbf{Z}}2l))/2,\\
&(x+k)/4+l=(x+(k+_{\mathbf{Z}}4l))/4,\\
&(x+k)/2+(y+l)/2=((x+y)+(k+_{\mathbf{Z}}l))/2,\\
&x+0=x, \qquad 0+y=y,\\
&0/2=0, \qquad 2k/2=k, \qquad k+l=k+_{\mathbf{Z}}l.
\end{aligned}
$$

It is crucial to treat everything connected with ρ (a type variable) as non-computational. This is a point where the non-computational quantifiers are essential.

The functions J and D. In the proof of lemma YSatClause we had to solve $d'+e'+2i=j+4d$ for given $d',e'\in\mathbf{SD}$ and $i\in\mathbf{SD}_2$. This is a finite problem and hence can be solved by defining $J:\mathbf{SD}\to\mathbf{SD}\to\mathbf{SD}_2\to\mathbf{SD}_2$ and $D:\mathbf{SD}\to\mathbf{SD}\to\mathbf{SD}_2\to\mathbf{SD}$ explicitly. The validity of $d'+e'+2i=J(d',e',i)+4D(d',e',i)$ is then verified by means of case distinctions.

3.2. *Extraction*

We present terms extracted from the formalized proofs (literal output of Minlog), and give some explanations.

Extraction from lemma XSubY. The term extracted from the proof is

```
[v0,v1]
 [if (des v0)
   [if (des v1)
     (MT@v0@v1)
     ([dv2]JOne M left dv2@v0@right dv2)]
   ([dv2]
    [if (des v1)
      (JOne left dv2 M@right dv2@v1)
      ([dv3]JOne left dv2 left dv3@right dv2@right dv3)])]
```

Here `v` is a name for variables ranging over $\mathbf{I}$, and `dv` for variables ranging over $\mathbf{SD}\times\mathbf{I}$. The constant `des` denotes the destructor for $\mathbf{I}$ of type $\mathbf{I}\to\mathbf{U}+\mathbf{SD}\times\mathbf{I}$, and `JOne`$:\mathbf{SD}\to\mathbf{SD}\to\mathbf{SD}_2$ adds the two integers. Clearly `left` and `right` are (prefix) operators for the components of a pair. The constant `cXSubY` of type $\mathbf{I}\to\mathbf{I}\to\mathbf{SD}_2\times\mathbf{I}\times\mathbf{I}$ is defined to be the term above. It satisfies the equations

$$
\begin{aligned}
&\mathtt{cXSubY}(\mathbb{I},\mathbb{I})=\langle 0,\mathbb{I},\mathbb{I}\rangle,\\
&\mathtt{cXSubY}(\mathbb{I},\mathrm{C}_e w)=\langle e,\mathbb{I},w\rangle,\\
&\mathtt{cXSubY}(\mathrm{C}_d v,\mathbb{I})=\langle d,v,\mathbb{I}\rangle,\\
&\mathtt{cXSubY}(\mathrm{C}_d v,\mathrm{C}_e w)=\langle d+e,v,w\rangle.
\end{aligned}
$$

https://doi.org/10.1017/S0960129513000327 Published online by Cambridge University Press

For the given two streams cXSubY computes the sum of the two head digits (regarding $\mathbb{I}$ as $\mathrm{C_M}\mathbb{I}$), and its tails. This sum of digits of type $\mathbf{SD}_2$ is a 'carry' which contains intermediate information to compute the average.

Extraction from lemma YSatClause. The term extracted from the proof is

```
[i0,v1,v2]
 [if (des v1)
   [if (des v2)
    (J M M i0@D M M i0@v1@v2)
    ([dv3]J M left dv3 i0@D M left dv3 i0@v1@right dv3)]
   ([dv3]
    [if (des v2)
      (J left dv3 M i0@D left dv3 M i0@right dv3@v2)
      ([dv4]J left dv3 left dv4 i0@D left dv3 left dv4 i0
            @right dv3@right dv4)])]
```

The constant cYSatClause of type $\mathbf{SD}_2 \to \mathbf{I} \to \mathbf{I} \to \mathbf{SD}_2 \times \mathbf{SD} \times \mathbf{I} \times \mathbf{I}$ is defined to be the term above. It satisfies the equations

$$\begin{aligned}
\mathtt{cYSatClause}(i, \mathbb{I}, \mathbb{I}) &= \langle J(0,0,i), D(0,0,i), \mathbb{I}, \mathbb{I}\rangle,\\
\mathtt{cYSatClause}(i, \mathbb{I}, \mathrm{C}_e w) &= \langle J(0,e,i), D(0,e,i), \mathbb{I}, w\rangle,\\
\mathtt{cYSatClause}(i, \mathrm{C}_d v, \mathbb{I}) &= \langle J(d,0,i), D(d,0,i), v, \mathbb{I}\rangle,\\
\mathtt{cYSatClause}(i, \mathrm{C}_d v, \mathrm{C}_e w) &= \langle J(d,e,i), D(d,e,i), v, w\rangle.
\end{aligned}$$

For the given carry and two signed digit streams, cYSatClause computes the carry for the next step, the first signed digit of the average of the streams, and the tails of the streams.

Extraction from theorem Average. The term extracted from the proof is

```
[v0,v1]
 (CoRec sdtwo@@iv@@iv=>iv)(cXSubY v0 v1)
 ([ivw2]
   Inr
   [let jdvw3
     (cYSatClause left ivw2 left right ivw2 right right ivw2)
     (left right jdvw3@
     (InR sdtwo@@iv@@iv iv)(left jdvw3@right right jdvw3))])
```

of type $\mathbf{I} \to \mathbf{I} \to \mathbf{I}$. It calls cXSubY to compute the first carry and the tails of the inputs. Then CoRec repeatedly calls cYSatClause in order to compute the average step by step. Here ivw is a name for variables ranging over $\mathbf{SD}_2 \times \mathbf{I} \times \mathbf{I}$, and jdvw for variables ranging over $\mathbf{SD}_2 \times \mathbf{SD} \times \mathbf{I} \times \mathbf{I}$. The second argument of the corecursion operator, say $M : \mathbf{SD}_2 \times \mathbf{I} \times \mathbf{I} \to \mathbf{U} + \mathbf{SD} \times (\mathbf{I} + \mathbf{SD}_2 \times \mathbf{I} \times \mathbf{I})$, operates on an argument $\langle i, v, w\rangle$ as follows. Let $\mathtt{cYSatClause}(i,v,w) = \langle j, d, v', w'\rangle$. Then $M\langle i,v,w\rangle = \mathrm{inr}\langle d, \mathrm{inr}\langle j, v', w'\rangle\rangle$. Given v and w, let $\mathtt{cXSubY}(v,w) = \langle i, v, w\rangle =: N$. Then MN is $\mathrm{inr}\langle d, \mathrm{inr}\langle j, v', w'\rangle\rangle$. Therefore by the conversion rule (6) for the corecursion operator the result is $\mathrm{C}_d({}^{\mathrm{co}}\mathcal{R}\langle j, v', w'\rangle M)$.

https://doi.org/10.1017/S0960129513000327 Published online by Cambridge University Press

The term above can be used as a program to compute the average for concrete input data. An experiment is described in Section 3.4. Comparing with a result by Berger and Seisenberger (2012), our extracted program behaves almost the same as theirs except for a difference coming from the data type of streams. They use the type of necessarily infinite streams, written $\mathrm{fix}\,\alpha.\mathrm{SD} \times \alpha$, whereas we accept possibly infinite streams (cototal ideals of $\mathbf{I}$).

3.3. *Normalization*

Recall that the term t extracted from a proof of a formula A is a realizer of A, in particular a term of the underlying formal theory TCF. This is in contrast to what is extracted in the proof assistants Coq (Letouzey 2003) and Isabelle (Berghofer 2003), which (for efficiency reasons) are programs in a programming language like OCaml, ML, Haskell or Scheme. We prefer to stay within TCF, for then the proposition $t\ \mathbf{r}\ A$ ('t realizes A') can be proved formally, as a special case of the soundness theorem. This gives a higher degree of security, since the formal proof of $t\ \mathbf{r}\ A$ can easily be machine checked, largely independent of the full proof assistant. When efficiency is an issue, one can – in a second step – translate the term t into an expression of a programming language and employ standard optimization techniques in the process. In Minlog the command is called `term-to-expr`; it has Scheme as the target language.

However, we can use the extracted (closed higher-order) term t directly as a program. For this to work we need to apply it to (closed) input or argument terms, and calculate (i.e. normalize) the resulting applicative term. For efficiency reasons we use normalization-by-evaluation (Berger *et al.* 2003) here. In case proofs involve coinduction the extracted term may contain corecursion operators. This creates a well-known problem for normalization, since the conversion rule (6) for the corecursion operator does not terminate. However, we can view this fact as a feature rather than a bug, since we *want* to compute with streams. In particular, we want to see the result only up to a given accuracy, which controls the number of unfoldings of the corecursion operator.

In the present case a simple approach suffices. First normalize the extracted term, treating the corecursion operator as a constant (i.e. without conversion rules), and then provide an external bound for the number of unfoldings of the corecursion operator. This could be refined to a more demand driven device, as Haskell's `take` command.

3.4. *An experiment*

Let `eterm` be the term above, extracted from the Average theorem. We want to use this term as a program to compute the average of $\frac{5}{8}$ and $\frac{3}{4}$. To this end we proceed as follows.

1. Normalize `eterm` to `neterm`, treating the corecursion operator as a constant.
2. Represent the two arguments as terms of type $\mathbf{I}$. Recall that the constructors of the algebra $\mathbf{I}$ are $\mathbb{I} : \mathbf{I}$ and $\mathrm{C} : \mathbf{SD} \to \mathbf{I} \to \mathbf{I}$, and that the three (nullary) constructors of $\mathbf{SD}$ are $-1, 0, 1$ (or internally `L`, `M`, `R` for left, middle, right); we write $\mathrm{C}_d v$ for $\mathrm{C}dv$. Then $\frac{5}{8} = \frac{1}{2} + \frac{1}{8}$ appears as $\mathrm{C}_{\mathtt{R}}(\mathrm{C}_{\mathtt{M}}(\mathrm{C}_{\mathtt{R}}\mathbb{I}))$ and $\frac{3}{4} = \frac{1}{2} + \frac{1}{4}$ as $\mathrm{C}_{\mathtt{R}}(\mathrm{C}_{\mathtt{R}}\mathbb{I})$.

https://doi.org/10.1017/S0960129513000327 Published online by Cambridge University Press

3. Let `test` be the result of applying `neterm` to the two arguments, and `ntest` its normal form. This term still contains the corecursion operator; it denotes a cototal ideal in **I** (i.e. a stream).
4. Normalize `ntest` again, but this time allowing say 10 unfoldings of the corecursion operator. The commands are

```
(define eterm10 (undelay-delayed-corec ntest 10))
(define neterm10 (nt eterm10))
(pp neterm10)
```

The final result is

```
C R (C R (C M (C L (C M (C M (C M (C M (C M (C M
                    ((CoRec sdtwo@@iv@@iv=>iv)...)))))))))
```

This is the correct result, for

$$\frac{\frac{5}{8}+\frac{3}{4}}{2}=\frac{11}{16}=\frac{1}{2}+\frac{1}{4}-\frac{1}{16}.$$

Since normalization of terms `nt` is implemented by means of normalization-by-evaluation, we have a reasonable efficiency: it takes 50 ms to evaluate `(nt eterm10)`.

4. Conclusion

We presented a formal proof of the existence of the average of two real numbers in $[-1, 1]$, as a case study in constructive exact real arithmetic. From the formal proof involving coinduction we extracted a term containing the corecursion operator. The evaluation method of the corecursion operator allows to approximate the computation of cototal ideals or streams, the standard representation of non-well-founded data as real numbers.

As for the future work, general uniformly continuous functions can be studied with the same motivation as in the present paper. A promising treatment is in Berger (2009) (or Schwichtenberg and Wainer (2012)) which involves nested algebras in TCF and their cototal/total ideals.

References

Agda. (2013) Available at http://wiki.portal.chalmers.se/agda/

Berger, U. (1993) Program extraction from normalization proofs. In: Bezem, M. and Groote, J. (eds.) Typed Lambda Calculi and Applications. *Springer Verlag Lecture Notes in Computer Science* **664** 91–106.

Berger, U. (2009) From coinductive proofs to exact real arithmetic. In: Grädel, E. and Kahle, R. (eds.) Computer Science Logic. *Springer Verlag Lecture Notes in Computer Science* **5771** 132–146.

Berger, U., Eberl, M. and Schwichtenberg, H. (2003) Term rewriting for normalization by evaluation. *Information and Computation* **183** 19–42.

Berger, U. and Seisenberger, M. (2010) Proofs, programs, processes. In: Ferreira, F *et al.* (eds.) Proceedings CiE 2010. *Springer Verlag Lecture Notes Computer Science* **6158** 39–48.

Berger, U. and Seisenberger, M. (2012) Proofs, programs, processes. *Theory of Computing* **51** 313–329.

https://doi.org/10.1017/S0960129513000327 Published online by Cambridge University Press

Berghofer, S. (2003) *Proofs, Programs and Executable Specifications in Higher Order Logic*, Ph.D. thesis, Institut für Informatik, TU München.

Chuang, C. M. (2011) *Extraction of Programs for Exact Real Number Computation Using Agda*, Ph.D. thesis, Swansea University, Wales, UK.

Ciaffaglione, A. and Gianantonio, P. D. (2006) A certified, corecursive implementation of exact real numbers. *Theoretical Computer Science* **351** 39–51.

Coq Development Team (2009) *The Coq Proof Assistant Reference Manual – Version 8.2*. Inria.

Kohlenbach, U. (2008) *Applied Proof Theory: Proof Interpretations and Their Use in Mathematics*, Springer Verlag, Berlin, Heidelberg.

Kreisel, G. (1959) Interpretation of analysis by means of constructive functionals of finite types. In: Heyting, A. (ed.) *Constructivity in Mathematics*. North-Holland, Amsterdam, 101–128.

Letouzey, P. (2003) A new extraction for coq. In: Geuvers, H. and Wiedijk, F. (eds.) Types for Proofs and Programs, Second International Workshop, TYPES 2002. *Springer Verlag Lecture Notes in Computer Science* **2646** 200–219.

O'Connor, R. (2009) *Incompleteness & Completeness. Formalizing Logic and Analysis in Type Theory*, Ph.D. thesis, Nijmegen University.

Plume, D. (1998) *A Calculator for Exact Real Number Computation*, Ph.D. thesis, University of Edinburgh.

Schwichtenberg, H. (2006) Minlog. In: Wiedijk, F. (ed.) The Seventeen Provers of the World. *Springer Verlag Lecture Notes in Artificial Intelligence* **3600** 151–157.

Schwichtenberg, H. (2008) Realizability interpretation of proofs in constructive analysis. *Theory of Computing Systems* **43** (3) 583–602.

Schwichtenberg, H. and Wainer, S. S. (2012) *Proofs and Computations*. Perspectives in Logic.Cambridge University Press.

Tucker, J. V. and Zucker, J. I. (1992) Theory of computation over stream algebras, and its applications. In: Havel, I. M. and Koubek, V. (eds.) Mathematical Foundations of Computer Science. *Springer Verlag Lecture Notes in Computer Science* **629** 62–80.

Wiedmer, E. (1977) *Exaktes Rechnen mit reellen Zahlen und anderen unendlichen Objekten*, Ph.D. thesis, ETH Zürich.

Wiedmer, E. (1980) Computing with infinite objects. *Theoretical Computer Science* **10** 133–155.

https://doi.org/10.1017/S0960129513000327 Published online by Cambridge University Press